Cplant*

Rolf Riesen Ron Brightwell Lee Ann Fisk Tramm Hudson Jim Otto
Arthur B. Maccabe
{rolf|bright|lafisk|tbhudso|jotto}@cs.sandia.gov and maccabe@cs.unm.edu
http://www.cs.sandia.gov/cplant

May 17, 1999

Abstract

The Computational Plant project at Sandia National Laboratories is developing a large-scale, massively parallel computing resource from a cluster of commodity computing and networking components. We are combining the knowledge and research of previous and ongoing commodity cluster projects with our expertise in designing, developing, using, and maintaining large-scale MPP machines.

This paper describes the main parts of the architecture and discusses the most important design choices and decisions. Scaling to hundreds and thousands of nodes requires more than simply combining readily-available software and hardware. We will highlight some of the more crucial pieces that make Cplant scalable.

1 Introduction

The Computational Plant (Cplant) project began at Sandia National Laboratories in Spring of 1997. The initial idea was to explore commodity-based cluster technology and build a facility that would provide compute cycles like a power plant provides electricity. Distant sites should be able to connect to a Cplant and receive compute cycles as easily as an appliance plugged into a wall socket receives electricity.

Since commodity-based technology changes at a rapid pace, Cplant should be able to incorporate new hardware quickly. When new generations of hardware become available, Cplant should grow to incorporate this new hardware. Older pieces that have been part of Cplant for three or four years should be pruned just as easily. The biological metaphor is meant to indicate that a Cplant can keep up with the technology curve much easier than the massively parallel processors (MPP) of the past.

These are lofty goals. In order to be successful and deliver something usable early on, we decided on the following approach. Cplant would borrow some concepts of the DOE ASCI Red Tflops machine [5] and initially have the feel of a large MPP. Our group designed the compute node operating system for Tflops and we wanted to bring its Puma Portals [6] to Cplant. At the same time, we wanted to add the capability to grow. To achieve this with our small group, we have chosen, for now, to keep the system as homogeneous as possible. This means new growth for Cplant consists of the same CPU architecture and the same network technology as the original prototype we built in 1997. Work has begun to allow Cplant to become more heterogenous, but much remains to be done.

The rest of this paper shows what we have accomplished and gives a little insight into how Cplant works and how it is built. Then, we will discuss some of the design decisions that we have made and explain why we made them. Finally, we outline some of the future work we have planned.

2 Cplant Architecture

Logically, a Cplant consists of several partitions [2]. A partition is a collection of nodes that together perform one of the functions of the total computational resource. These partitions are defined in configuration files, and the boundaries can be moved by reconfiguring or rebooting the nodes in the affected partitions. When users logon to a Cplant, a load-balancing name server puts them onto one of the nodes in the **service partition**, where users interact with Cplant and launch parallel applications. The size of the service partition (i.e., the number of nodes assigned to the service partition) is determined by the expected number of simultaneous users. For a few users running large parallel applications, one or a few service

*This work was performed at Sandia National Laboratories and supported by the United States Department of Energy under Contract DE-AC04-94AL8500.

nodes will do. If there are many users running many small jobs, then the service partition needs to be bigger.

Most of the nodes in a Cplant are in the **compute partition**, where parallel application processes run. Users cannot logon to these nodes, and we keep the number of services running on compute nodes to a minimum. Ideally, only application processes consume CPU cycles on a compute node. We are even thinking of trimming Linux down to a very simple kernel that cannot do much more than message passing and run a compute intensive process. Our compute nodes have no local disks attached to them. For that matter, none of the nodes in Cplant have keyboards, video cards, CD-ROMs, or floppy drives. The exception is the top level system support station (discussed below) that controls the whole system.

A utility called *yod* is used in the service partition to launch a parallel application in the compute partition. This launch utility and a set of daemons allocate compute nodes and place one process per compute node. For the duration of the run, these compute nodes remain assigned to this parallel application. Nodes are thus space-shared. Time-sharing compute nodes would require demand-paged virtual memory. The overhead associated with context switches and page swapping would be disastrous to our applications which need a lot of physical memory to run at full speed. When demand increases, and not enough compute nodes are available, then the system is probably too small for the number of applications and simultaneously required nodes. At that time the Cplant can be grown to a larger size.

There are other partitions. I/O partitions consist of nodes that have disks attached. We are working on a parallel file system that converts I/O from compute nodes into messages and accesses the data stored on the disks attached to the nodes in the I/O partition. The nodes of any partition can be anywhere in the topology; partitioning is just a logical arrangement enforced by our software. The node allocator tries to select compute nodes that are physically close together. This strategy reduces message passing latency for applications and also reduces congestion in the network. One strategy for I/O nodes is to distribute them evenly across the topology. If compute nodes select I/O nodes nearby, then I/O traffic is evenly distributed across the network. However, it is not always possible to chose a nearby I/O node. For example, a data transpose requires data to be read to a different compute node than the one that wrote it. In our configuration, the I/O nodes are not the same PCs as the compute nodes. They have different dimensions and we installed them in their own racks. They share one Myrinet switch and are therefore located in one spot of the network. The switch has enough bisection bandwidth and enough external connections into the network that this setup should not impact performance.

Other partitions can be created. For example, nodes with long-distance network interfaces, such as ATM, can be collected in a network partition. Nodes with special graphics hardware could be combined in a graphics partition.

Another key aspect of the Cplant architecture is maintainability. Our current 400-node system uses Myrinet to connect all the service, compute, and I/O nodes. We have built a support infrastructure that is hidden from the users. It is a crucial piece of a large-scale system. Yet, just like the internals of a personal computer, users need not be aware of all the functions involved in making the system usable and reliable.

We call this support infrastructure the system support partition, even though it has nothing in common with the configurable partitions mentioned earlier. The system support partition consists of separate, Ethernet based, networks and a set of PCs. These PCs, called system support stations (sss), are not connected to the Myrinet, have a disk with the system software on it, and serve as the controllers for a scalable unit (SU). An SU in our current configuration consists of 16 nodes (compute, service, or I/O) collected in two racks together with power controllers, Myrinet switches, terminal servers, and an Ethernet hub. The system support station controls the power to all the devices that are part of its SU, serves kernels and firmware to the devices in its SU, allows telnet to the console ports (through the terminal server), and provides NFS mounts for the system software.

All the system support stations at level 0 are connected through yet another Ethernet, to a level 1 system support station. This sss1 is the Cplant entry point for system administrators and developers. From here all devices in the system can be power-cycled, booted, and monitored. This is also the point where the system software gets installed. Scripts using *rdist* distribute the software to the system support stations at level 0. This hierarchy of support stations is very scalable and adaptable. Each individual Ethernet has a minimal load on it, no matter how big the system grows. Using this support system we can power cycle and boot a 400-node Cplant in under two minutes.

3 System Software

The system software can be grouped into several major parts. Portals and the associated drivers are our low-level message passing mechanism. All the traffic going over Myrinet uses Portals. Libraries, such as MPI and I/O, use Portals to transfer data and control messages.

The application loader *yod* works in conjunction with a node allocator and a daemon process on each compute node to load and start application processes. Special startup code, executed before `main()`, sets up the environment each process sees and establishes a connection to *yod*. The loader remains running in the service partition as long as the parallel application is active. After startup, *yod* serves as a proxy. Signals issued to *yod* are propagated to all processes in the application. Standard I/O from any process in the application gets funneled through *yod*. A `printf()` to standard out will appear on the terminal from which *yod* has been started. The environment an application process sees is the environment *yod* sees on the service node. For example, environment variables such as $USER, are visible from each process. A home directory mounted on the service node and accessible by *yod*, is automatically accessible by all the processes that *yod* launched.

Funneling all I/O through *yod* and a single service node is not scalable. For this reason we are implementing a parallel file system that will use multiple data paths to multiple I/O nodes and disks to distribute the load. *Yod* and its functionality can be easily integrated into a batch queuing system such as PBS.

Another major piece of the system software can be found in the system support partition. Here we have configuration files, a hardware database, and scripts. We can specify what nodes belong to what partition, which kernels should be booted and what version of the system software should be used on a given node. We can boot and power cycle individual nodes, an SU, or the whole system from the top level system support station. This sss1 is on the Sandia network. Therefore, we can control operation of Cplant from our offices and even from home. We have scripts that automatically discover hardware when the system is powered on. With the information gained during that process we build a database of available hardware. A set of rules is used by perl scripts to automatically generate configuration files that determine such things as the host names of nodes, what kernels they should boot, what services need to be started, and what partition they should belong to.

4 Design Decisions

In this section we discuss some of our design decisions. The overriding design goal for Cplant is the ability to scale to at least 8192 nodes. Developing and choosing software and hardware for that scale ensures that smaller machines will run without problems. This requirement also excludes right from the start many options that work well on 32 or even 64 nodes.

Why Linux? We had several options when we started looking for an operating system for Cplant. We considered porting Puma, the compute node OS from the Tflops machine. We thought about using the vendor supplied DEC Unix, and we looked at several of the free Unix clones. We needed to come up fast and we also needed a full-fledged Unix in the service partition. This eliminated porting Puma. Given source access, DEC Unix was an option. We chose Linux because it is open source and has been ported to many different architectures that might some day become a part of Cplant. Becoming independent of a single vendor is one of Cplant's goals. Therefore, we need an OS that runs on many platforms and is vendor neutral.

Another project at Sandia already had Linux running on individual nodes of the Tflops machine, and a port of Portals to Linux had begun.

Among the free Unix clones we chose Linux because we already had some familiarity using it on our desktop machine and its kernel modules provided for a quick and easy way to develop and test kernel enhancements. Linux is now the dominant free Unix. Many vendors support it on their platforms and a large users and developers community ensures longevity.

Why Portals? Portals are openings into user address space [6] that can be written to or read from. The process that opens a Portal has several options to select access to that Portal, and how incoming messages should be deposited. A key feature is the ability to select incoming messages based on their source, length, and a 64-bit tag. This selection takes place before the message is deposited in user space and helps avoid costly memory-to-memory copies.

We developed Portals on the nCUBE 2, ported them to the Intel Paragon, and they are in daily use on the Tflops machine. We have libraries, such as MPI, that make use of Portals, and they have proven to be scalable.

We looked at other options, such as AM II, GAM, BIP, FM, GM, PM, VIA, and ST. We created a communication layer requirements document for Cplant. This layer has to support MPI for applications, but also I/O, process loading and control, and debugging of remote processes. Cplant is designed to use a wide variety of network technologies, and the communication layer has to support all of them. All other choices for this layer were eliminated because they were specific to a given network technology, not backed by a major vendor, not scalable, or too early in their development to be usable in a production environment.

Why Myrinet? We wanted the fastest, scalable network available. This ideal was constrained by cost. Myrinet is a mature gigabit network technology. Its switches can be connected in many different way to build any desired topology. Myrinet has its roots in the networks of large MPPs and is scalable. Last but not least, the interface cards are programmable and allow system programmers to implement various protocols directly in the interface cards.

Why a custom loader? A simple shell script that spawns *rsh* processes is not scalable beyond a few hundred nodes. We already had a scalable application process loader on the Tflops machine. It also included a node allocator, standard I/O redirection, signal propagation, and other desirable features.

Why no local disk? We are buying standard desktop PCs as nodes for our Cplant. However, we configure them to exclude CD-ROM, floppy drive, keyboard, mouse, video card, and hard-disk. The first reason to exclude a local hard-disk is DOE laboratory specific. In order to use a system for classified and unclassified work, any permanent storage has to be removed when switching from one mode of operation to another. With two separate file servers, one attached to Cplant when the other is not, we have the option of switching the compute nodes quickly from classified to unclassified mode.

We do not want to use disks to demand-page memory. Our high-performance applications would suffer significantly, if required pages need to be made resident before a message can be deposited or sent. This means the only purpose for a local disk is scratch space or as a component of a parallel file server. Using local scratch space has the disadvantage that it is tied to a particular node. After a checkpoint restart, or as input to a subsequent application, local scratch space cannot be used because the application may run on a different set of nodes. Instead of moving the data to the new nodes, it is more economical to use a parallel file system that is location independent as scratch space.

Why not SMP? A second CPU on each node would add only about 1/3 more compute power using Linux. Furthermore, Linux is still not a very good SMP OS. Its stability decreases in SMP mode. There are other reasons not to chose SMP nodes just yet. Our libraries, including our modified MPICH, are not thread-safe. The programming model becomes more complex and most of our applications are not ready to take advantage of other local CPUs.

A second CPU could be used as a message co-processor. This is not as effective as it is on the Paragon and the Tflops machine, because the network interface is on the PCI bus, not the memory bus. Since the Myrinet cards already provide a programmable CPU, using a host CPU as a message co-processor is not cost effective.

5 Status

The system is running Release 0.2 of our system software. Friendly users have been using the system for several months, and an effort is under way to bring about ten new applications to Cplant this Summer. The goal is to run each of these applications on at least 256 nodes.

The integration of Puma Portals from the small runtime kernel of the Tflops machine into Linux proved challenging. The fact that the network interface is behind an I/O bus, instead of directly on the memory bus, and dealing with a full-fledged OS kernel caused unwanted overhead in our implementation, such as memory-to-memory copies.

We have redesigned Portals, simplified them, added a thin API, and created a reference implementation. The reference implementation is modular and is divided into the Portals 3.0 proper and a network dependent piece. The reference implementation comes with a network abstraction layer that sits on top of TCP/IP. The first implementation over Myrinet replaces the current Portals module in the Linux kernel. Much of the overhead is still present, since we reuse the drivers and Myrinet control program (MCP) on the network interface card (NIC) from the current implementation. We have begun to port our system software and Myrinet library to the Portals 3.0 API.

While adaptation to the new API is taking place, we have also begun work on the second implementation of Portals 3.0. For this implementation we put the Portals code into the Myrinet NIC. The CPU on the NIC can then deposit and retrieve data from user memory without the intervention of the host CPU. For this OS bypass mechanism to be efficient, we need physically contiguous memory under Linux. This is one area where Linux does not fit our needs and we had to modify the kernel.

The 400-node Cplant is operational and in daily use. Almost 800 more nodes have been ordered and will be installed later this Summer. The new system employs nodes with the DEC Alpha 21264 (EV6), more memory (256MB), and faster Myrinet cards (64-bit LANai 7.x). The nodes will be divided into several smaller Cplants. Some of them to do classified work, and some of them to go to the Sandia California site.

A major piece of 400 to 500 nodes will remain and will be brought up independently at first, but will then be connected to the original 400 node Cplant, making it one of the largest clusters in the world. And it runs Linux!

6 Conclusion and Future Work

Our approach to building a Linux cluster is strongly influenced by our experience writing system software for the nCUBE 2, Intel Paragon, and DOE ASCI Red Tflops machine. We have built a very large cluster, paying close attention to scalability, dealing with issues that never show up on 128 nodes or less. We wrote the system software so that the cluster would feel, and can be managed, like a single large MPP.

We think there is a spectrum of systems ranging from widely distributed systems to tightly coupled MPPs [4]. Each serves its purpose. Our users at Sandia National Laboratories are used to MPPs and have applications that are well adapted to such an environment. These applications are tuned to run on hundreds and thousands of nodes. To serve our community, we created software that mimics that environment using standard off-the-shelf components.

In the future we will continue to grow Cplant to truly large scales and attempt to characterize how large of a system, useful to MPP applications, can be built using off-the-shelf components. Cplant will become more heterogenous and we will have to deal with the challenges this presents. Another group at Sandia has begun looking at making Cplant compute resources available to other facilities in the DOE complex. This group is looking at Globus [1] and Legion [3] to see how these systems could be used to make Cplant a node in wide-area compute resources.

7 Acknowledgements

The system software team for Cplant includes the authors and Bill Davidson, Michael Levenhagen, Mike McConkey, John VanDyke and Lee Ward. Many people outside of this core group have contributed and we would like to thank Rob Armstrong, Pang Chen, Robert Clay, David Evensky, David Greenberg, Mark Sears, and Dave van Dresser.

References

[1] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufman, 1998.

[2] David S. Greenberg, Ron Brightwell, Lee Ann Fisk, Arthur B. Maccabe, and Rolf Riesen. A system software architecture for high-end computing. In *Proceedings of Supercomputing'97*, San Jose, CA, November 1997.

[3] Andrew Grimshaw, Adam Ferrari, Frederick Knabe, and Marty Humphrey. Wide-area computing: Resource sharing on a large scale. *IEEE Computer*, 32(5):29–37, May 1999.

[4] Rolf Riesen, Ron Brightwell, and Arthur B. Maccabe. Differences between distributed and parallel systems. Technical report SAND98-2221, Sandia National Laboratories, 1998.

[5] Tom Thompson. The world's fastest computers. *Byte*, 21(1):45–64, January 1996.

[6] Stephen R. Wheat, Arthur B. Maccabe, Rolf Riesen, David W. van Dresser, and T. Mack Stallcup. PUMA: An operating system for massively parallel systems. *Scientific Programming*, 3:275–288, 1994.